University of Freiburg
Dept. of Computer Science
Prof. Dr. F. Kuhn
P. Bamberger, P. Schneider

Algorithm Theory
Sample Solution Exercise Sheet 2

Due: Tuesday, 2nd of November, 2021, 4 pm

Exercise 1: Computing the Median *(10 Points)*

Let A be an *unsorted* Array of *pairwise distinct* integers of length n. We want to compute the median of A, i.e., the element $m \in A$ that would be in the middle of A if we would sort A (we say the median is the smaller of the two "middle" elements in case A is of even length). We want to accomplish this *deterministically*[1] in time $O(n)$.

Remark: You can not assume that the size of integers in A is constant in n, thus simply sorting A is not possible in $O(n)$ time.

(a) We start with an algorithm that computes a value relatively close to the median. The first step is to partition the elements of A into $k := \lceil \frac{n}{5} \rceil$ consecutive sub-arrays (group) A_i ($i \in \{1, \dots, k\}$) of 5 elements each (the last group A_k may be smaller). Then compute the median m_i of each group A_i. Let m' be the median of $m_1, \dots, m_k$. Show that at least $\frac{3n}{10}$ elements in A are smaller than or equal to m' *and* $\frac{3n}{10}$ elements in A are larger than or equal to m'. (Edit: a previous version made the claim for the smaller fraction $\frac{n}{5}$ instead of $\frac{3n}{10}$, but the proof is basically the same.) *(3 Points)*

Hint: You may assume that n is divisible by 5.

(b) Give a divide and conquer algorithm to compute the j^{th}-largest element of A in time $O(n)$ for some j (edit: or analogously compute the j^{th}-smallest, either can be used to compute the median). Argue why your algorithm is correct and why it has the desired running time. *(7 Points)*

Hint: Use part (a) as subroutine.

Sample Solution

(a) Let $k' := \lceil k/2 \rceil$ be the index of the "group median" $m' = m_{k'}$ of $m_1, \dots, m_k$. Then the medians $m_1, \dots, m_{k'}$ are smaller than or equal to m' and $m_{k'}, \dots, m_k$ are larger than or equal to m'. In either case, these are at least $\lceil k/2 \rceil$ group medians which are smaller-equal or larger-equal m', respectively.

Since we assume all groups A_i are of size 5 (i.e., n is divisible by 5) for each group A_i with $m_i \geq m'$ at least 3 elements in A_i are larger-equal m'. That means in such a group a fraction of 3/5 of elements is larger-equal m'. Since the condition $m_i \geq m'$ holds for $\lceil k/2 \rceil$ many groups, i.e. at least half of them, we have that $\frac{1}{2} \cdot \frac{3}{5} \cdot n = \frac{3n}{10}$ elements are larger-equal m'. By symmetry, the same holds for the number of elements smaller-equal m'.

(b) *Remark: It is algorithmically of little consequence if we search for the j^{th}-smallest or j^{th}-largest element, as the j^{th}-smallest is obtained by computing the $(n-j+1)^{th}$-largest and vice versa.*

Assume we have a subroutine called `group-medians(A)` that returns an array containing the medians m_i of the groups A_i specified in part (a) together with their original indices in A (which

[1] That is, the algorithm must always succeed within the claimed running time.

we need to recover the index of m'). The runtime for this step is the same as iterating A once and every 5 steps attach the median of the last 5 elements to the output, i.e., $\mathcal{O}(n)$.

Further, we use the partition step known from Quicksort as a subroutine `partition(`A, p`)`. It rearranges the content A such that all elements smaller-equal $A[p]$ are to the left of position p and all elements larger than $A[p]$ are to the right of index p in A and returns the new position of $A[p]$ in the resulting array. This takes $\mathcal{O}(n)$ time.

The following routine `find(`j, A`)` computes the $(j{+}1)^{th}$-*smallest* element in A (since the array A is zero-based).

Algorithm 1 `find(`j, A`)` ▷ *assert* $j \in \{0, \ldots, n-1\}$

$n \leftarrow |A|$
if $n = 1$ **then** ▷ *base case*
  return $A[0]$
$B \leftarrow$ `group-medians(`A`)`
$k \leftarrow |B|$
$m' \leftarrow$ `find(`$\lceil \frac{k}{2} \rceil - 1, B$`)` ▷ *median of medians*
$p \leftarrow$ index of m' in A
$\ell \leftarrow$ `partition(`A, p`)` ▷ *elements smaller* $A[\ell]$ *left, larger* $A[\ell]$ *right,* $A[\ell]$ *in final position*
if $j = \ell$ **then** ▷ j^{th}*-smallest found*
  return $A[\ell]$
else if $j < \ell$ **then** ▷ j^{th}*-smallest must be in* $A[0..\ell{-}1]$
  return `find(`$j, A[0..\ell{-}1]$`)`
else ▷ j^{th}*-smallest must be in* $A[(\ell{+}1)..n]$
  return `find(`$j{-}(\ell+1), A[(\ell{+}1)..n]$`)`

Running time: A call of `find(`j, A`)` has a running time of $\mathcal{O}(n)$ to compute the group medians and do the partition, plus the runtime of the two recursive calls of the function. The first recursive call is on an instance of size roughly $n/5$.

The second recursive call is on a subarray $A[1..\ell-1]$ or $A[(\ell+1)..n]$. where ℓ is the index of m' after partitioning. We know that m' is larger-equal and smaller-equal $\frac{3n}{10}$ elements in A. This is therefore equal to the number of elements that we loose in subarrays $A[1..\ell-1]$ or $A[(\ell+1)..n]$ and therefore both are of size at most $\frac{7n}{10}$.

The function for the running time can thus be given recursively as $T(n) \leq T\left(\frac{n}{5}\right) + T\left(\frac{7n}{10}\right) + c \cdot n$ for some constant $c > 0$. We claim that $T(n) \leq 10 \cdot c \cdot n$. In the base case $n = 1$ this is certainly true (for an appropriate constant c) as we just make a check and immediately return a value. Inductively (hypothesizing that the claim is true for all $n' < n$) we get that

$$T(n) \leq T\left(\tfrac{n}{5}\right) + T\left(\tfrac{7n}{10}\right) + c \cdot n \overset{(Hypothesis)}{\leq} 10 \cdot c \cdot \left(\tfrac{n}{5} + \tfrac{7n}{10}\right) + c \cdot n = 10 \cdot c \cdot n.$$

Correctness: We make an inductive argument over n. If A has just $n = 1$ element we can clearly return $A[0]$. Presume correctness for all $n' < n$. After the partition step all elements smaller than $A(\ell)$ are to its left and all elements larger than $A(\ell)$ to its right and $A(\ell)$ is at the correct position it would also have if A were sorted. So if $j = \ell$ we can be certain that this is the j^{th}-smallest element and return it.

Else, if $j < \ell$, then the j^{th}-smallest element in A must be to the left of index ℓ, which is why we get the correct result with the recursive call on a strictly smaller subarray $A[1..\ell{-}1]$ (by induction hypothesis).

Else, if $j > \ell$ then the j^{th}-smallest element in A must be to the right of index ℓ. However, the j^{th}-smallest element in A now corresponds to the $(j-\ell-1)^{th}$-smallest element in $A[(\ell{+}1)..n]$, since we loose $\ell{+}1$ elements in $A[0..\ell]$. With this modified search index the recursive call `find(`$j{-}(\ell+1), A[(\ell{+}1)..n]$`)` returns the correct result (by induction hypothesis).

Exercise 2: Fast Fourier Transformation (FFT) *(10 Points)*

Let $p(x) = 8x^7 + 7x^6 + 6x^5 + 5x^4 + 4x^3 + 3x^2 + 2x + 1$. We want to compute the discrete fourier transform $DFT_8(p)$ (where we define $DFT_8(p) := DFT_8(a)$ given that a is the vector of coefficients of p). More specifically, we want you to visualize the steps which the FFT-algorithm performs as follows.

(a) Illustrate the *divide* procedure of the algorithm. More precisely, for the i-th divide step, write down all the polynomials p_{ij} for $j \in \{0, \dots, 2^i - 1\}$ that you obtain from further dividing the polynomials from the previous divide step $i-1$ (we define $p_{00} := p$). *(3 Points)*

(b) Illustrate the *combine* procedure of the algorithm. That is, starting with the polynomials of smallest degree as base cases, compute the $DFT_N(p_{ij})$ bottom up with the recursive formula given in the lecture (where N is the smallest power of 2 such that $\deg(p_{ij}) < N$). *(7 Points)*

Remarks: The base case for a polynomial $p = a$ of degree 0 is $DFT_1(p) = DFT_1(a) = a$. It suffices to give the $p_{ij}(\omega)$ for all N^{th} roots of unity ω, from which $DFT_N(p_{ij})$ can be derived. Use $\sqrt{\cdot}$ instead of floating point numbers if possible (for instance $\omega_8^1 = \frac{i+1}{\sqrt{2}}$ and $\omega_8^3 = \frac{i-1}{\sqrt{2}}$).

Sample Solution

(a)

$$
\begin{aligned}
p_{00}(x) &= 8x^7 + 7x^6 + 6x^5 + 5x^4 + 4x^3 + 3x^2 + 2x + 1 \\
p_{10}(x) &= 7x^3 + 5x^2 + 3x + 1 \\
p_{11}(x) &= 8x^3 + 6x^2 + 4x + 2 \\
p_{20}(x) &= 5x + 1 \\
p_{21}(x) &= 7x + 3 \\
p_{22}(x) &= 6x + 2 \\
p_{23}(x) &= 8x + 4 \\
p_{30}(x) &= 1 \\
p_{31}(x) &= 5 \\
p_{32}(x) &= 3 \\
p_{33}(x) &= 7 \\
p_{34}(x) &= 2 \\
p_{35}(x) &= 6 \\
p_{36}(x) &= 4 \\
p_{37}(x) &= 8
\end{aligned}
$$

(b) Base cases of the FFT algorithm (for any $x \in \mathbb{C}$):

$$
\begin{aligned}
p_{30}(x) &= DFT_1(p_{30}) = 1 \\
p_{31}(x) &= DFT_1(p_{31}) = 5 \\
p_{32}(x) &= DFT_1(p_{32}) = 3 \\
p_{33}(x) &= DFT_1(p_{33}) = 7 \\
p_{34}(x) &= DFT_1(p_{34}) = 2 \\
p_{35}(x) &= DFT_1(p_{35}) = 6 \\
p_{36}(x) &= DFT_1(p_{36}) = 4 \\
p_{37}(x) &= DFT_1(p_{37}) = 8
\end{aligned}
$$

Bottom up computation with the recursive formula:

$$\begin{aligned}
p_{20}(\omega_2^0) &= p_{30}(\omega_1^0) + \omega_2^0 \cdot p_{31}(\omega_1^0) = 1 + 1 \cdot 5 = 6 \\
p_{20}(\omega_2^1) &= p_{30}(\omega_1^0) - \omega_2^0 \cdot p_{31}(\omega_1^0) = 1 - 1 \cdot 5 = -4
\end{aligned}$$

$$\begin{aligned}
p_{21}(\omega_2^0) &= p_{32}(\omega_1^0) + \omega_2^0 \cdot p_{33}(\omega_1^0) = 3 + 1 \cdot 7 = 10 \\
p_{21}(\omega_2^1) &= p_{32}(\omega_1^0) - \omega_2^0 \cdot p_{33}(\omega_1^0) = 3 - 1 \cdot 7 = -4
\end{aligned}$$

$$\begin{aligned}
p_{22}(\omega_2^0) &= p_{34}(\omega_1^0) + \omega_2^0 \cdot p_{35}(\omega_1^0) = 2 + 1 \cdot 6 = 8 \\
p_{22}(\omega_2^1) &= p_{34}(\omega_1^0) - \omega_2^0 \cdot p_{35}(\omega_1^0) = 2 - 1 \cdot 6 = -4
\end{aligned}$$

$$\begin{aligned}
p_{23}(\omega_2^0) &= p_{36}(\omega_1^0) + \omega_2^0 \cdot p_{37}(\omega_1^0) = 4 + 1 \cdot 8 = 12 \\
p_{23}(\omega_2^1) &= p_{36}(\omega_1^0) - \omega_2^0 \cdot p_{37}(\omega_1^0) = 4 - 1 \cdot 8 = -4
\end{aligned}$$

$$\begin{aligned}
p_{10}(\omega_4^0) &= p_{20}(\omega_2^0) + \omega_4^0 \cdot p_{21}(\omega_2^0) = 6 + 1 \cdot 10 = 16 \\
p_{10}(\omega_4^1) &= p_{20}(\omega_2^1) + \omega_4^1 \cdot p_{21}(\omega_2^1) = -4 + i \cdot (-4) = -4 - 4i \\
p_{10}(\omega_4^2) &= p_{20}(\omega_2^0) - \omega_4^0 \cdot p_{21}(\omega_2^0) = 6 - 1 \cdot 10 = -4 \\
p_{10}(\omega_4^3) &= p_{20}(\omega_2^1) - \omega_4^1 \cdot p_{21}(\omega_2^1) = -4 - i \cdot (-4) = -4 + 4i
\end{aligned}$$

$$\begin{aligned}
p_{11}(\omega_4^0) &= p_{22}(\omega_2^0) + \omega_4^0 \cdot p_{23}(\omega_2^0) = 8 + 1 \cdot 12 = 20 \\
p_{11}(\omega_4^1) &= p_{22}(\omega_2^1) + \omega_4^1 \cdot p_{23}(\omega_2^1) = -4 + i \cdot (-4) = -4 - 4i \\
p_{11}(\omega_4^2) &= p_{22}(\omega_2^0) - \omega_4^0 \cdot p_{23}(\omega_2^0) = 8 - 1 \cdot 12 = -4 \\
p_{11}(\omega_4^3) &= p_{22}(\omega_2^1) - \omega_4^1 \cdot p_{23}(\omega_2^1) = -4 - i \cdot (-4) = -4 + 4i
\end{aligned}$$

$$\begin{aligned}
p_{00}(\omega_8^0) &= p_{10}(\omega_4^0) + \omega_8^0 \cdot p_{11}(\omega_4^0) = 16 + 1 \cdot 20 = 36 \\
p_{00}(\omega_8^1) &= p_{10}(\omega_4^1) + \omega_8^1 \cdot p_{11}(\omega_4^1) = -4 - 4i + \tfrac{i+1}{\sqrt{2}} \cdot (-4 - 4i) = -4 - 4i \cdot (\sqrt{2}+1) \\
p_{00}(\omega_8^2) &= p_{10}(\omega_4^2) + \omega_8^2 \cdot p_{11}(\omega_4^2) = -4 + i \cdot (-4) = -4 - 4i \\
p_{00}(\omega_8^3) &= p_{10}(\omega_4^3) + \omega_8^3 \cdot p_{11}(\omega_4^3) = -4 + 4i + \tfrac{i-1}{\sqrt{2}} \cdot (-4 + 4i) = -4 - 4i \cdot (\sqrt{2}-1) \\
p_{00}(\omega_8^4) &= p_{10}(\omega_4^0) - \omega_8^0 \cdot p_{11}(\omega_4^0) = 16 - 1 \cdot 20 = -4 \\
p_{00}(\omega_8^5) &= p_{10}(\omega_4^1) - \omega_8^1 \cdot p_{11}(\omega_4^1) = -4 - 4i - \tfrac{i+1}{\sqrt{2}} \cdot (-4 - 4i) = -4 + 4i \cdot (\sqrt{2}-1) \\
p_{00}(\omega_8^6) &= p_{10}(\omega_4^2) - \omega_8^2 \cdot p_{11}(\omega_4^2) = -4 - i \cdot (-4) = -4 + 4i \\
p_{00}(\omega_8^7) &= p_{10}(\omega_4^3) - \omega_8^3 \cdot p_{11}(\omega_4^3) = -4 + 4i - \tfrac{i-1}{\sqrt{2}} \cdot (-4 + 4i) = -4 + 4i \cdot (\sqrt{2}+1)
\end{aligned}$$

Rewriting the discrete fourier transforms as vectors (not strictly necessary, though):

$$\begin{aligned}
DFT_2(p_{20}) &= (6, -4) \\
DFT_2(p_{21}) &= (10, -4) \\
DFT_2(p_{22}) &= (8, -4) \\
DFT_2(p_{23}) &= (12, -4)
\end{aligned}$$

$$\begin{aligned}
DFT_4(p_{10}) &= (16, -4 - 4i, -4, -4 + 4i) \\
DFT_4(p_{11}) &= (20, -4 - 4i, -4, -4 + 4i)
\end{aligned}$$

$$\begin{aligned}
DFT_8(p_{00}) = \big(&36, -4 - 4i \cdot (\sqrt{2}+1), -4 - 4i, -4 - 4i \cdot (\sqrt{2}-1), \\
&-4, -4 + 4i \cdot (\sqrt{2}-1), -4 + 4i, -4 + 4i \cdot (\sqrt{2}+1)\big)
\end{aligned}$$